Published in the Proceedings of 2000 Supercomputing Conference (SC '00).

The Failure of TCP in High-Performance Computational Grids*

W. Feng[§†] and P. Tinnakornsrisuphap[‡]
feng@lanl.gov, tinnakor@cae.wisc.edu

[§] Research & Development in Advanced Network Technology (RADIANT)
P.O. Box 1663, M.S. B255
Los Alamos National Laboratory
Los Alamos, NM 87545

[†] Department of Computer & Information Science
Ohio State University
2015 Neil Avenue
Columbus, OH 43210

[‡] Department of Electrical & Computer Engineering
University of Wisconsin-Madison
1415 Engineering Drive
Madison, WI 53706

Abstract

Distributed computational grids depend on TCP to ensure reliable end-to-end communication between nodes across the wide-area network (WAN). Unfortunately, TCP performance can be abysmal even when buffers on the end hosts are manually optimized. Recent studies blame the self-similar nature of aggregate network traffic for TCP's poor performance because such traffic is not readily amenable to statistical multiplexing in the Internet, and hence computational grids.

In this paper, we identify a source of self-similarity previously ignored, a source that is readily controllable — TCP. Via an experimental study, we examine the effects of the TCP stack on network traffic using different implementations of TCP. We show that even when aggregate application traffic ought to smooth out as more applications' traffic are multiplexed, TCP induces burstiness into the aggregate traffic load, thus adversely impacting network performance. Furthermore, our results indicate that TCP performance will worsen as WAN speeds continue to increase.

Keywords: network traffic characterization, self-similarity, TCP, computational grid, distributed computing

*This work was supported by the U.S. Dept. of Energy through Los Alamos National Laboratory contract W-7405-ENG-36. This paper is LA-UR 00-3765.

1 Introduction

Distributed computational grids, managed by middleware such as Globus [4] and Legion [5], require support for the fluctuating and heterogeneous demands of individual users. The ability to characterize the behavior of the resulting aggregate network traffic can provide insight into how traffic should be scheduled to make efficient use of the network in a computational grid.

Several studies conclude that aggregate network traffic is self-similar (or fractal) [7, 12], and thus not amenable to the statistical-multiplexing techniques currently found on the Internet. Additional studies claim that the heavy-tailed distributions of file size, packet interarrival, and transfer duration fundamentally contribute to the self-similar nature of aggregate network traffic [10, 11, 16]. While these heavy-tailed distributions may contribute to self-similarity, we will illustrate that TCP itself is the primary source of self-similarity and that this behavior may have dire consequences in computational grids as WAN speeds increase into the gigabit-per-second (Gb/s) range. In particular, we show that even when application traffic is rigged to produce a Hurst parameter H of 0.5, TCP adversely modulates this

application traffic and produces $H = 1.0$ (i.e., self-similar traffic). While we focus on the TCP-induced self-similarity of TCP Reno and TCP Vegas, concurrent research by Veres et al. [14, 15] proves that TCP Tahoe is also a source of self-similarity but does so through an examination of congestion windows ($cwnd$) and their attractors.

2 Background

TCP is a connection-oriented service which guarantees the reliable, in-order delivery of a stream of bytes, hence freeing the application from having to worry about missing or reordered data. It includes a flow-control mechanism which ensures that a sender does not overrun the buffer capacity of the receiver and a congestion-control mechanism which tries to prevent too much data from being injected into the network, thereby causing packet loss within the network. While the size of the flow-control window is static, the size of the congestion window evolves over time, according to the status of the network.

2.1 TCP Congestion Control

Currently, the most widely-used TCP implementation is TCP Reno [6]. Its congestion-control mechanism consists of two phases: (1) slow start and (2) congestion avoidance. In the slow-start phase, the congestion window grows exponentially (i.e., doubles every time the sender successfully transmits a congestion-window's worth of packets across the network) until a timeout occurs, which implies that a packet has been lost. At this point, a $Threshold$ value is set to the halved window size; TCP Reno resets the congestion window size to one and re-enters the slow-start phase, increasing the congestion window exponentially up to the $Threshold$. When the threshold is reached, TCP Reno then enters its congestion-avoidance phase in which the congestion window is increased by "one packet" every time the sender successfully transmits a congestion-window's worth of packets across the network. When a packet is lost during the congestion-avoidance phase, TCP Reno takes the same actions as when a packet is lost during slow start.

TCP Reno now also implements fast-retransmit and fast-recovery mechanisms for both the slow-start and congestion-avoidance phases. Rather than timing out while waiting for the acknowledgement (ACK) of a lost packet, if the sender receives three duplicate ACKs (indicating that some packet was lost but later packets were received), the sender immediately retransmits the lost packet (fast retransmit). Because later packets were received, the network congestion is assumed to be less severe than if all packets were lost, and the sender only halves its congestion window and re-enters the congestion-avoidance phase (fast recovery) without going through the slow-start phase again.

TCP Vegas [1] introduces a new congestion-control mechanism that tries to prevent rather than react to congestion. When the congestion window increases in size, the expected sending rate (ER) increases as well. But if the actual sending rate (AR) stays roughly the same, this implies that there is *not* enough bandwidth available to send at ER, and thus, any increase in the size of the congestion window will result in packets filling up the buffer space at the bottleneck gateway. TCP Vegas attempts to detect this phenomenon and avoid congestion at the bottleneck gateway by adjusting the congestion-window size, and hence ER, as necessary to adapt to the available bandwidth.

To adjust the window size appropriately, TCP Vegas defines two threshold values, α and β, for the congestion-avoidance phase, and a third threshold value, γ, for the transition between the slow-start and congestion-avoidance phases. Conceptually, $\alpha = 1$ implies that TCP Vegas tries to keep at least one packet from each stream queued in gateway while $\beta = 3$ keeps at most three packets.

If *Diff = ER - AR*, then when *Diff* $< \alpha$, Vegas increases the congestion window linearly during the next RTT; when *Diff* $> \beta$, Vegas decreases the window linearly during the next RTT; otherwise, the congestion window remains unchanged. The γ parameter can be viewed as the "initial" β when TCP Vegas enters its congestion-avoidance phase.

To further enhance TCP performance, Floyd et al. proposed the use of random early detection (RED) gateways [3] to detect incipient congestion. RED gateways maintain a weighted average of the queue length. As long as the average queue length stays below the minimum threshold (min_{th}), all packets are queued. When the average queue length exceeds min_{th}, packets are dropped with probability P. And when the average queue length exceeds a maximum threshold (max_{th}), all arriving packets are dropped.

2.2 Probability and Statistics

The Central Limit Theorem states that the sum of a large number of finite-mean, finite-variance, independent variables (e.g., Poisson) approaches a Gaussian random variable with less variability (i.e., less "spread" or burstiness) than the original distribution(s). So, if each random variable represented traffic generated by a particular communication stream, then the sum of a large number of these streams represents aggregate network traffic with less variability, and thus less variation or spread in the required bandwidth, i.e, network traffic is less bursty or more smooth. Such aggregate traffic behavior enables statistical-multiplexing techniques to be effective over the Internet. Unfortunately, even if application-generated traffic streams have finite means and variances and are independent, TCP can modulate these streams in such a way that they are no longer independent, leading to aggregate network traffic with wildly oscillating

and unpredictable bandwidth demands [13].

To measure the burstiness of aggregate TCP traffic, we use the *coefficient of variation (c.o.v.)* [2, 13] — the ratio of the standard deviation to the mean of the observed number of packets arriving at a gateway in each round-trip propagation delay. The $c.o.v.$ gives a normalized value for the "spread" of a distribution and allows for the comparison of "spreads" over a varying number of communication streams. If the $c.o.v.$ is small, the amount of traffic coming into the gateway in each RTT will concentrate mostly around the mean, and therefore will yield better performance via statistical multiplexing. For purposes of comparison, we also use the *Hurst parameter*, H, from self-similar modeling [7, 10, 11, 12, 16]. While H may be better at determining long-term buffer requirements (i.e., large time granularities), it does not provide insight into how well statistical multiplexing performs, i.e., at small time granularities such as a round-trip time (RTT).

3 Experimental Study

To understand the dynamics of TCP, we use *ns* [9] to model the portion of the Energy Sciences network (ESnet) that is specific to our computational grid — namely the network between Los Alamos National Laboratory (LANL) and Sandia National Laboratory (SNL). We then synthetically generate application-generated traffic to better understand how TCP modulates input traffic. Understanding *how* TCP modulates traffic can have a profound impact on the $c.o.v.$ and H parameters, and hence, the throughput and packet loss percentage of network traffic.

3.1 Network Topology

Figures 1 and 2 show the current ESnet backbone and an abstraction of the ESnet backbone, respectively. We assume that there are up to M hosts in the local-area networks (LANs) of both LANL and SNL sending traffic back and forth. The traffic generated from an Ethernet-attached host i in LANL's LAN goes through an Ethernet/FDDI interface to a FDDI/ATM Cisco 7507 router to a Fore ASX-200BX ATM switch in Albuquerque (ABQ POP) to a Cisco 7507 router to host i at SNL and vice versa. While the traffic generated between LANL and SNL alone cannot cause any congestion at the ABQ POP's WAN ATM switch, data from the "Internet Cloud" is significant enough to interfere with traffic from both sites.

The Cisco 7507 routers have buffers large enough (48 MB) to handle incoming and outgoing traffic. The ABQ POP's Fore ASX-200BX switch can only buffer 64K ATM cells per port or about 3 MB per port.

Figure 3 shows an abstraction of the proposed topology for the future ESnet backbone. The ABQ POP ATM switch remains a Fore ASX-200BX, but the interface speed changes to OC-12 (622 Mb/s). The routers on either side of the ATM switch upgrade to Cisco 7512 routers which have 33% more buffer space than the 7507s. Lastly, both LAN architectures are replaced by Gigabit Ethernet (1000 Mb/s). In short, all the link bandwidths scale up, but the ABQ POP ATM switch remains essentially the same.

3.2 Traffic Characterization

The $c.o.v.$ provides a quantitative measure of how traffic smoothes out when a large number of finite-mean, finite-variance, independent streams are aggregated (via the Central Limit Theorem). To characterize the TCP modulation of traffic, we first generate application traffic according to a known distribution, e.g., Poisson. We then compare the $c.o.v.$ of this distribution (i.e., theoretical $c.o.v.$) to the $c.o.v.$ of the traffic transmitted by TCP (i.e., measured $c.o.v.$). This allows us to determine whether TCP modulates the traffic, and if so, how it affects the shape (burstiness) of the traffic, and hence the performance of the network.

For instance, when N independent Poisson sources each generate packets at a rate of λ with round-trip time delay of τ, the $c.o.v.$ of the aggregated sources is

$$\text{c.o.v.(Poisson,}\tau) = \frac{1}{\sqrt{\lambda N \tau}} \qquad (1)$$

Hence, this theoretical $c.o.v.$ of Poisson traffic decreases as a function of $1/\sqrt{N}$, implying that traffic should become smoother as N increases (or as more sources are aggregated).

The self-similar model is also used to characterize network traffic. In this model, the Hurst parameter H theoretically lies between 0.5 and 1.0. When $H = 1.0$, the aggregate traffic is self-similar and exhibits long-range dependence. Because we generate application traffic for each client identically according to a known distribution, i.e., Poisson, with a finite mean and variance, the Hurst parameter H from self-similar modeling should be 0.5 *if* TCP does not adversely modulate the application traffic. However, as we will show later in this section, TCP modulates the traffic to have self-similar characteristics, particularly TCP Reno.

3.3 Experimental Set-Up

Tables 1 and 2 show the network parameters for the current and proposed ESnet topologies, respectively. The values for network delay are derived from *traceroute* information between LANL and SNL. Network-hardware values such as buffer size are derived from vendor specifications. Tables 3 and 4 contain appropriately scaled traffic-source parameters for the current and proposed ESnet, respectively.

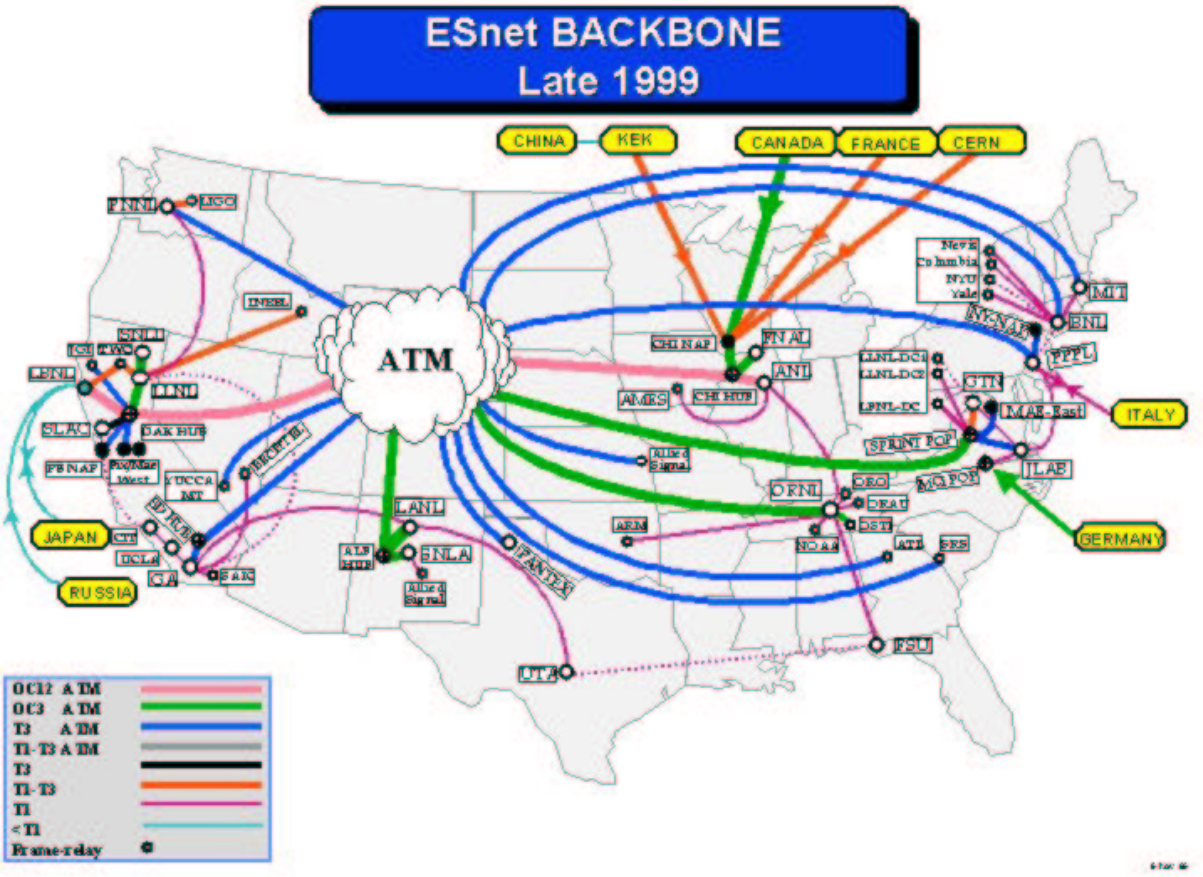

Figure 1. Network Topology of the Energy Sciences Network (ESNet)

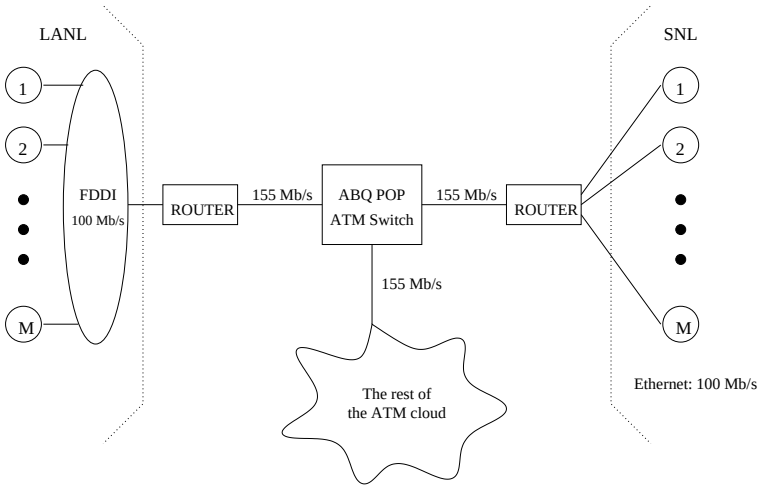

Figure 2. Abstraction of the Current ESnet Architecture.

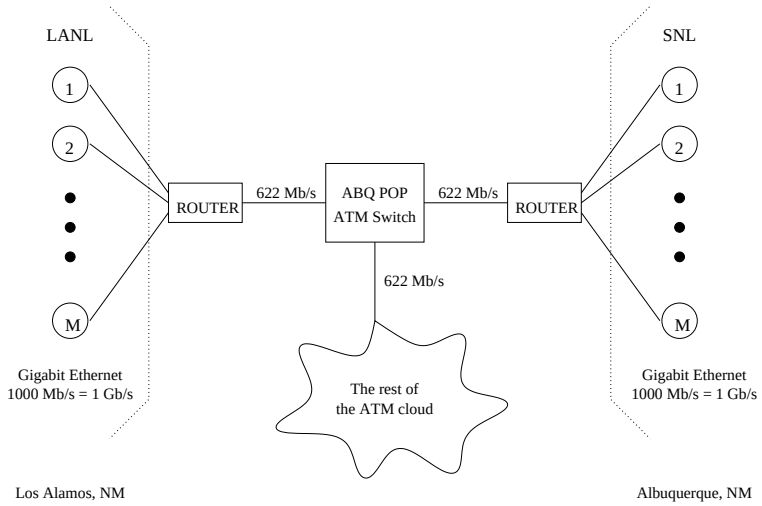

Figure 3. Abstraction of the Proposed ESnet Architecture.

Each LANL client generates Poisson traffic, i.e., single packets are submitted to the TCP stack with exponentially distributed interpacket arrival times with mean $1/\lambda$. We vary the total traffic load offered by varying the number of clients M and test two different implementations of TCP (Reno and Vegas) and two different queueing disciplines in the routers (FIFO and RED). The cross traffic produced by the rest of the "ATM Cloud" is modeled after a Poisson session arrival with Pareto service time for each session. For each session, a new TCP is randomly created to either LANL or SNL. This traffic can be characterized by the arrival rate, mean file size, and Pareto shape parameter.

To see how TCP modulates traffic, we calculate the theoretical $c.o.v.$ of the aggregate traffic generated by the LANL clients (based on the Poisson distribution each client uses to generate its traffic) and compare it to the measured $c.o.v.$ of the aggregate TCP-modulated traffic at the routers.

For example, using the experimental parameters from Table 1, the theoretical $c.o.v.$ of Poisson traffic is 0.659 when $N = 2$ and drops all the way down to 0.120 when $N = 60$. However, in Sections 4 and 5, we will find that the measured $c.o.v.$ (after TCP modulation) can be upwards of 300% larger than the theoretical $c.o.v.$!

4 Results: Current ESnet Topology

When the amount of traffic being generated is larger than the available bandwidth, i.e., from Table 3, $\lceil (155 - 64)\ Mb/s\ /\ 3\ Mb/s/host \rceil = 31$ hosts, TCP congestion-control kicks in and modulates the traffic to be more bursty than expected, i.e., the measured $c.o.v.$ is significantly larger than the theoretical $c.o.v.$ from Eq. 1, as shown in Figure 4. In particular, with the theoretical $c.o.v.$ reaching a low of 0.120 when $N = 60$, the measured $c.o.v.$ for TCP Reno is generally 300% higher than the theoretical $c.o.v.$ while the measured $c.o.v.$ for TCP Vegas is never more than 40% higher (and in most cases is less than 10% higher).

TCP Reno begins to induce burstiness when as few as eight clients are aggregated. By 20 clients, the induced burstiness becomes so significant that it adversely impacts both throughput and packet-loss rates, as shown in Figures 5 and 6. (These results are in stark contrast to our results in a more commodity 10-Mb/s switched Ethernet environment [13].) The enhanced burstiness that we see here results from a combination of (1) the fluctuation of the congestion-window sizes, particularly in TCP Reno [13], and (2) the magnification of TCP's inability to adapt appropriately to short-lived congestion over high bandwidth-delay links.

Figure 5 shows the number of packets transmitted through both routers. The throughput becomes saturated around 30 clients or when the link capacity minus the average throughput of outside sources equals the traffic generated by the sources. TCP Vegas obviously does a better job at utilizing the link as the number of packets transmitted is higher than in TCP Reno. This substantiates the findings in [8].

Figure 6 shows that TCP Vegas does not lose any packets, i.e., packet-loss rate = 0, because its α and β parameters effectively avoid congestion. In contrast, TCP Reno loses a measurable 0.26% of packets when the network is not even congested at less than 20 clients. The 0.26% loss rate corroborates the loss-rate numbers reported by ESnet; while such a number is oftentimes dismissed, it should not be because when a packet is dropped before it reaches its destination, all the resources that it has consumed in transit are wasted. Over a 155 Mb/s link, a loss rate of 0.26%

Parameter	Value
LANL network speed	100 Mb/s
LANL network delay	0.3385 ms
LANL FDDI to ATM interface delay	0.3315 ms
LANL to ABQ POP speed	155 Mb/s
LANL to ABQ POP delay	1.21 ms
LANL router buffer size	33554 packets
SNL network speed	100 Mb/s
SNL network delay	0.287 ms
SNL to ABQ POP speed	155 Mb/s
SNL to ABQ POP delay	0.13 ms
SNL router buffer size	33554 packets
ABQ POP ATM buffer (per port)	2315 packets
Outside traffic link speed	155 Mb/s
Outside traffic link delay	10 ms
TCP max advertised window	10000 packets
Packet size	1500 bytes
Round-trip propagation delay	4.6 ms
Total test time	200 s
TCP Vegas/α, β, γ	1, 3, 1
RED gateway/min_{th}	6711 packets
RED gateway/max_{th}	26843 packets

Table 1. Network Parameters for Current ESnet.

Parameter	Value
LANL network speed	1000 Mb/s
LANL network delay	0.67 ms
LANL to ABQ POP speed	622 Mb/s
LANL to ABQ POP delay	1.21 ms
LANL router buffer size	44739 packets
SNL network speed	1000 Mb/s
SNL network delay	0.287 ms
SNL to ABQ POP speed	622 Mb/s
SNL to ABQ POP delay	0.13 ms
SNL router buffer size	44739 packets
ABQ POP ATM buffer (per port)	2315 packets
Outside traffic link speed	622 Mb/s
Outside traffic link delay	10 ms
TCP max advertised window	100000 packets
Packet size	1500 bytes
Round-trip propagation delay	4.6 ms
Total test time	200 s
TCP Vegas/α, β, γ	1, 3, 1
RED gateway/min_{th}	8948 packets
RED gateway/max_{th}	35791 packets

Table 2. Network Parameters for Proposed ESnet.

Parameters	Value
Maximum # of clients	60
Poisson mean packet intergeneration ($1/\lambda$)	4 ms
Average traffic rate	3 Mb/s
ON/OFF mean burst period	3 ms
ON/OFF mean idle period	97 ms
ON/OFF Pareto shape	1.5
ON traffic rate	100 Mb/s
Average traffic rate	3 Mb/s
Outside-source mean time between session	0.25 s
Outside-source mean file size per session	2 MB
Outside-source average traffic rate	64 Mb/s

Table 3. Traffic-Source Parameters for Current ESnet.

Parameters	Value
Maximum # of clients	60
Poisson mean packet intergeneration ($1/\lambda$)	0.8 ms
Average traffic rate	15 Mb/s
ON/OFF mean burst period	1.5 ms
ON/OFF mean idle period	98.5 ms
ON/OFF Pareto shape	1.5
ON traffic rate	1000 Mb/s
Average traffic rate	15 Mb/s
Outside-source mean time between session	0.25 s
Outside-source mean file size per session	2 MB
Outside-source average traffic rate	64 Mb/s

Table 4. Traffic-Source Parameters for Proposed ESnet.

translates to 403 Kb of information being lost per second. Furthermore, this situation gets worse as the WAN scales up in speed (see Section 5 for details).

Figures 5 and 6 also show that the presence of a RED gateway at both the LANL and SNL routers does not cause much change in the $c.o.v.$ or the overall network performance because the bottleneck of the network is at the ABQ POP, not at the routers. In addition, because the buffer size in the ATM switch at the ABQ POP is small compared to both routers (which have buffers that are several orders of magnitude higher than the bandwidth-delay product), the buffers occupied in the routers are almost always smaller than the RED min_{th}, resulting in very little difference in network performance or the $c.o.v.$ of aggregate traffic.

Figure 7 shows the Hurst parameter H as a function of traffic load. If $H = 0.5$, the traffic streams are independent of one another; while at $H = 1.0$, the traffic streams exhibit self-similarity (burstiness), or more precisely, long-range dependence. So, when N independent Poisson sources are mathematically aggregated together, the Hurst parameter H is 0.5. If TCP does not adversely modulate our application-generated traffic of independent Poisson sources, H should remain at 0.5 across all offered traffic loads.

However, as Figure 7 shows, TCP Reno dramatically induces burstiness and long-range dependence into the traffic streams as its H value quickly reaches 1.0 when only twelve clients have been aggregated together over an uncongested network. Meanwhile, TCP Vegas does a much better job at not adversely modulating the traffic. Unfortunately, TCP Reno (not Vegas) is currently the most popular and virtually ubiquitous implementation of TCP out in the world today.

5 Results: Proposed ESnet Topology

As in the previous section, when the amount of traffic being generated is larger than the available bandwidth, i.e., $\lceil (622 - 64)/15 \rceil = 38$ hosts, in the proposed ESnet, TCP congestion-control really kicks in and adversely modulates the application traffic. However, over this proposed ESnet, the adverse modulation by TCP is even more pronounced than in the current ESnet, i.e., the measured $c.o.v.$ is enormously larger than the theoretical $c.o.v.$ from Eq. 1, as shown in Figure 8. In this particular case, the theoretical $c.o.v.$ reaches a low of 0.054 when $N = 60$ while the measured $c.o.v.$ for TCP Reno and TCP Vegas are upwards of 642% and 457% higher than theoretical $c.o.v.$! These numbers indicate that TCP performance may worsen as WAN speeds continue to increase; further evidence of this is provided below.

TCP Reno starts to induce burstiness when as few as eight clients are aggregated. By 34 clients, the induced burstiness becomes so significant that it adversely impacts both throughput and packet-loss rates, as shown in Figures 9

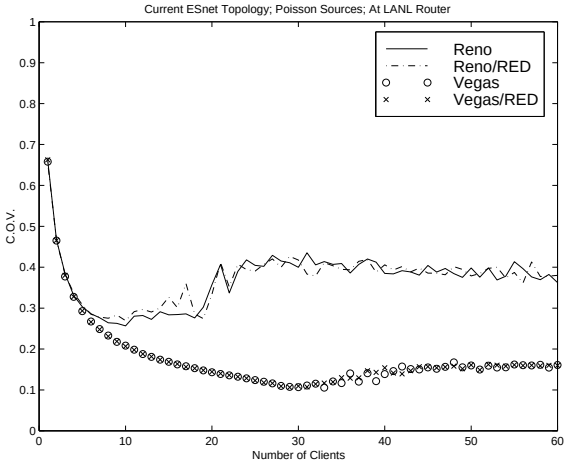

Figure 4. C.O.V. of Traffic at LANL Router.

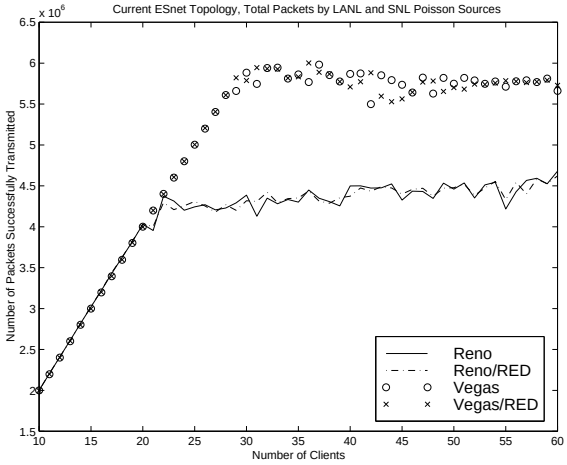

Figure 5. Number of Packets Transmitted Through LANL and SNL Routers.

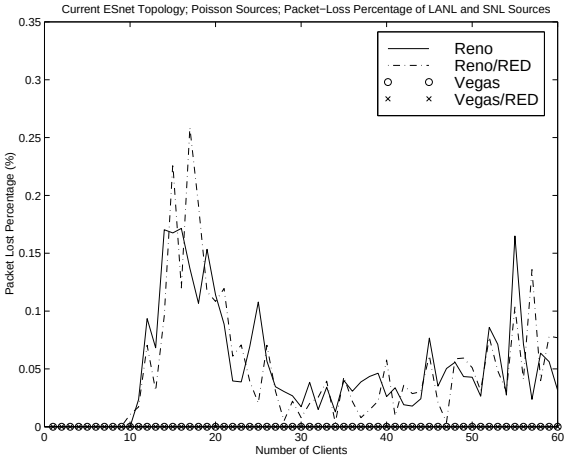

Figure 6. Packet-Loss Percentage for LANL and SNL Sources.

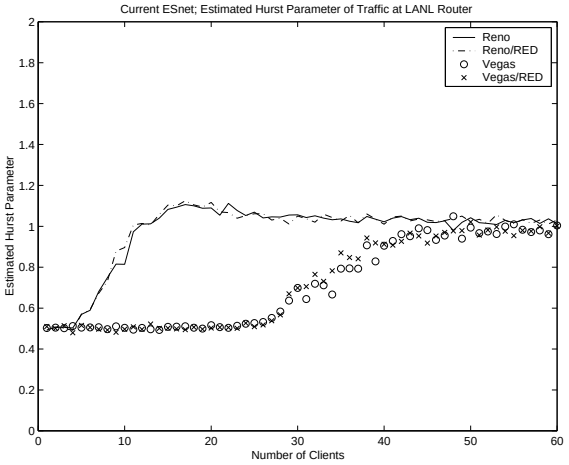

Figure 7. Hurst Parameter at LANL Router.

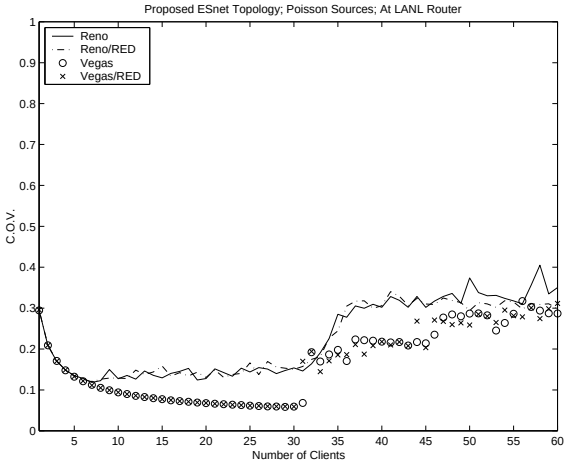

Figure 8. C.O.V. of Traffic at LANL Router.

and 10.

In contrast to Figure 5 where TCP Vegas shows better throughput than TCP Reno, Figure 9 shows that the throughput of TCP Reno is slightly better than TCP Vegas. However, a detailed look at a snapshot of the congestion-window evolution (Figures 11- 14) indicates that the reason why the throughput of TCP Reno is better than TCP Vegas may be due to the many packet retransmissions by TCP Reno. Consequently, TCP Vegas's goodput is likely to be better than TCP Reno's.

Figures 11 and 13 show that TCP Reno increases its congestion window to an extraordinarily high value ($>> 1000$) during slow start even though the optimal value is less than 100 packets. Why does this happen? Because there is no mechanism other than packet loss to inform TCP Reno of the appropriate value for the congestion window. With the large bursts that TCP Reno allows with the O(1000 packet) congestion window, the network becomes severely congested and must back off twice in Figure 13 without transmitting anything for more than two seconds. This suggests that the slow-start mechanism of TCP Reno does not adapt quickly enough when the bandwidth-delay product is very large.

Returning to figure on packet-loss percentage (Figure 10), TCP Reno's loss rate exceeds 5% when the network is heavily congested while TCP Vegas once again produces no packet loss. Over a 622 Mb/s link a loss rate of 5% translates to a loss of over 31 Mb/s! This kind of loss rate is clearly unacceptable for the multimedia applications that must be supported in computational grids, e.g., remote steering of visualization data and video-teleconferencing.

As in Section 4, Figures 9 and 10 also show that the presence of a RED gateway at both the LANL and SNL routers does not cause much change in the $c.o.v.$ or the overall net-

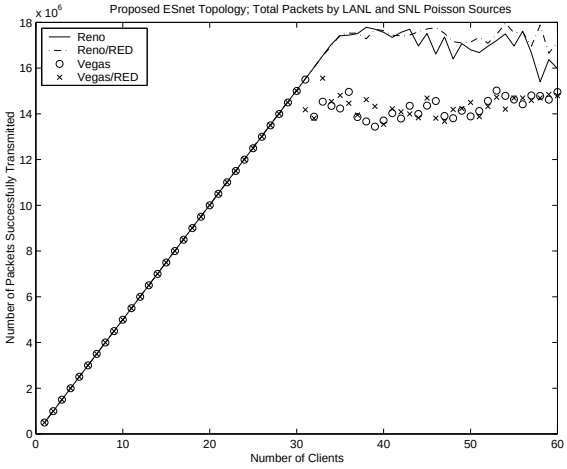

Figure 9. Number of Packets Transmitted Through LANL and SNL Routers.

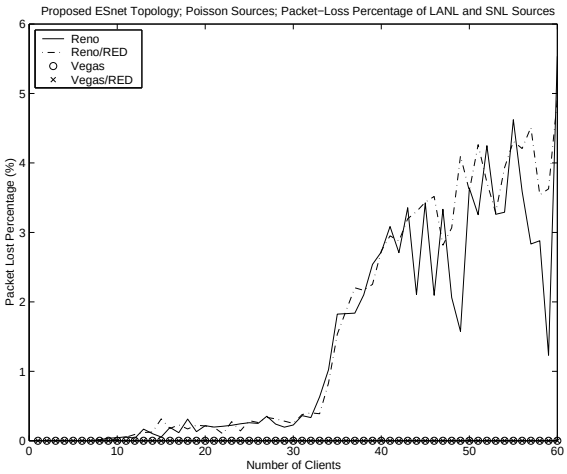

Figure 10. Packet-Loss Percentage for LANL and SNL Sources.

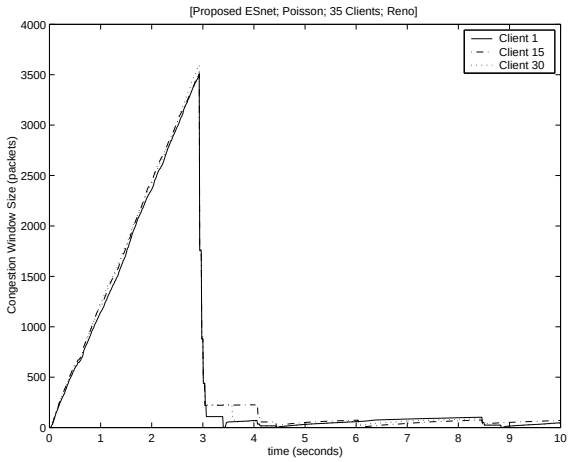

Figure 11. TCP Reno Congestion Window (Clients = 35).

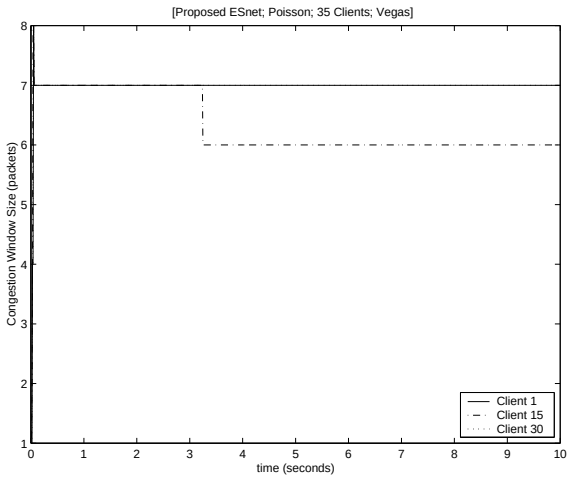

Figure 12. TCP Vegas Congestion Window (Clients = 35).

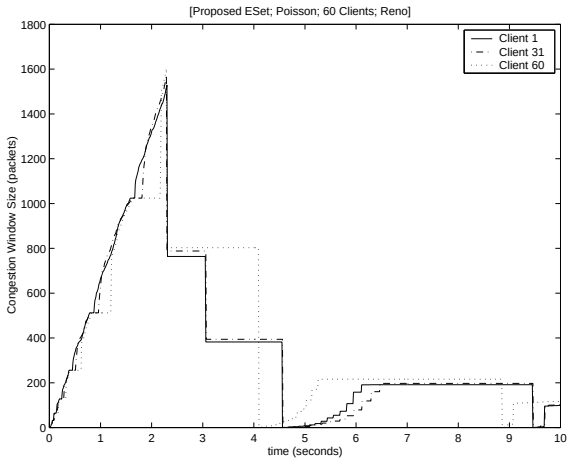

Figure 13. TCP Reno Congestion Window (Clients = 60).

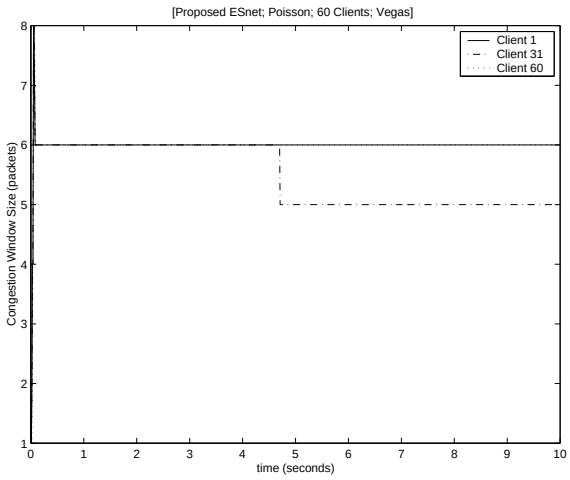

Figure 14. TCP Vegas Congestion Window (Clients = 60).

work performance because the bottleneck of the network is at the ABQ POP, not at the routers. In addition, because the buffer size in the ATM switch at the ABQ POP is small compared to both routers (which have buffers that are several orders of magnitude higher than the bandwidth-delay product), the buffers occupied in the routers are almost always smaller than the RED min_{th}, resulting in very little difference in network performance or the $c.o.v.$ of aggregate traffic.

6 Discussion

What are the implications of the above results for tomorrow's Next-Generation Internet and high-performance computational grids? First, in contrast to the literature over the past five years, we have shown that TCP itself is a primary source of self-similarity, particularly TCP Reno. Due to the burstiness that is induced by TCP Reno, the throughput and packet-loss metrics are adversely affected, reducing overall network performance.

Second, based on our results, Vegas [1] deserves serious consideration as a high-performance TCP for both the Internet as well as distributed computational grids. (However, the weakness in this claim is that we did not test TCP Vegas in a dynamically changing environment of short-lived connections. That is, we did not model the arrival and departure patterns of TCP connections.) For a somewhat opposing view, the authors direct the reader to [8] where Mo et al. pit a TCP Reno connection against a TCP Vegas connection with the input traffic originating from an infinite file stream.

Third, the performance of TCP *worsens* as WAN speeds are scaled up. Evidence of this was presented in Section 5. In addition, there exists another case where TCP performance will suffer as WAN speeds scale; a case that was not specifically addressed in our experimental study. For an intuitive understanding, one must think about the TCP Reno congestion-control mechanism in the context of a high bandwidth-delay product, e.g., 1 Gb/s WAN $\times$ 100 ms round-trip time (RTT) = 100 Mb. For the sake of argument, assume that the "optimal window size" for a particular connection is 50 Mb. TCP Reno continually increases its window size until it induces packet loss (i.e., just above 50 Mb) and then chops its window size in half (i.e., 25 Mb). Thus, having all TCP connections use packet loss as a way to incessantly probe network state can obviously induce burstiness. Furthermore, re-convergence to the "optimal window size" using TCP's absolute linear increase takes much too long and results in lowered network utilization. In this particular case, convergence can take as long as (50 Mb - 25 Mb) / (1500 B/RTT * 8 b/B) = 2,084 RTTs or (2,084 RTTs * 100 ms/RTT) = 208.4 seconds = 3.472 minutes.

7 Conclusion

The ability to characterize the behavior of aggregate network traffic can provide insight into how traffic should be scheduled to make efficient use of the network, and yet still deliver expected quality-of-service to end users. These issues are of fundamental importance in widely-distributed, high-speed computational grids.

Our experimental study illustrates that the congestion-control mechanisms of TCP Reno and TCP Vegas modulate the traffic generated by the application layer. TCP Reno, in particular, adversely modulates the traffic to be significantly more bursty, which subsequently affects the performance of statistical multiplexing in the gateway. This modulation occurs for a number of reasons: (1) the rapid fluctuation of the congestion-window sizes caused by the continual "additive increase / multiplicative decrease (or re-start slow start)" probing of the network state and (2) the dependency between the congestion-control decisions made by multiple TCP streams, i.e., TCP streams tend to recognize congestion in the network at the same time and thus halve their congestion windows at the same time (see Figures 11 and 13, for example).

Furthermore, over the proposed ESnet topology, we were able to magnify TCP's inability (particularly Reno) to adapt appropriately to congestion over high bandwidth-delay links. Thus, the work presented here concludes that if we continue on the path that we are on — using TCP Reno as the networking substrate for high-performance computational grids — overall network performance will suffer. In particular, TCP Reno's congestion-control mechanism induces burstiness and dependency between streams which ultimately limit the effectiveness of statistical multiplexing in routers. In addition, TCP Reno's "packet-loss induced" probing of network state does not allow a TCP connection to maintain its optimal window size.

Finally, this work also opens up future research opportunities in the profiling of application-generated network traffic. Why is such profiling important? Because the traffic distribution that sources generate will likely have an impact on the performance of the congestion-control protocol (even though the average aggregated traffic rate is the same). While the research community currently understands what network traffic looks like on the wire, there is little understanding on what the traffic looks like when it enters the TCP protocol. Oftentimes, researchers simply use an infinite-sized file as input into TCP. Why is this bad? Because it is analogous to pumping an infinitely long, memory-reference pattern into a cache-coherency protocol in order to test the effectiveness of the protocol. Rather than do that, researchers in computer architecture profiled memory-reference patterns in real parallel programs and used these profiles as input into their cache-coherency pro-

tocols. Likewise, we, the network research community, ought to start doing the same.

References

[1] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal of Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[2] W. Feng and P. Tinnakornsrisuphap. The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems. In *Proceedings of the International Conference on Parallel Processing (ICPP'00)*, August 2000.

[3] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[4] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman Publishing, 1999.

[5] A. Grimshaw, W. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[6] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM'88 Symposium*, pages 314–332, August 1988.

[7] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transaction on Networking*, 2(1):1–15, Feburary 1994.

[8] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proceedings of INFOCOM'99*, March 1999.

[9] ns. Network Simulator. *http://www-mash.cs.berkeley.edu/ns*.

[10] K. Park, G. Kim, and M. Crovella. On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic. In *Proceedings of the 4th International Conference on Network Protocols*, October 1996.

[11] K. Park, G. Kim, and M.Crovella. On the Effect of Traffic Self-Similarty on Network Performance. In *Proceedings of the SPIE International Conference on Performance and Control of Network Systems*, 1997.

[12] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transaction on Networking*, 3(3):226–244, June 1995.

[13] P. Tinnakornsrisuphap, W. Feng, and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'00)*, April 2000.

[14] A. Veres and M. Boda. The chaotic nature of tcp congestion control. In *Proceedings of INFOCOM 2000*, March 2000.

[15] A. Veres, Zs. Kenesi, S. Molnar, and G. Vattay. On the propagation of long-range dependence in the internet. In *Proceedings of SIGCOMM 2000*, August/September 2000.

[16] W. Willinger, V. Paxson, and M. Taqqu. Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, pages 27–53, 1998.